Redux y Thunk

En este commit se va a implementar Redux para la gestión de estados.

1. Configurando Redux en nuestra aplicación

Como primer paso se han instalado los paquetes necesarios para poner en marcha Redux con los siguientes comandos:

yarn add redux
yarn add react-redux
yarn add redux-thunk
yarn add redux-logger

A continuación, se va a crear la carpeta redux en el directorio aquí se insertará todos los archivos de redux.

Configurar el Store

En el fichero *configureStore.js* se define la función *ConfigureStore(),* será la responsable de crear el store de Redux y de combinar los cuatro *reducers*. Se va a optar por crear un repositorio diferente para cada uno de los tipos de datos que queremos almacenar (excursiones, cabecera, comentarios y actividades) que posteriormente se combinarán con la función *combineReducers*. Además, también asociará a dicho store los dos *middleware* que emplearemos en nuestro Redux: Thunk y Logger.

Action Types (tipos de acciones)

Se define todos los tipos de acciones asociados a nuestro repositorio Redux, en el fichero ActionTypes.js. Se define varios tipos de acciones para cada uno de los 4 *reducers* que componen el store. Acciones como: añadir información (ADD), saber si la información está siendo cargada (LOADING) o si la operación no se ha sido completada (FAILED).

Reducers

Aquí se construye los cuatro repositorios Redux que darán soporte a la aplicación (*excursiones.js, cabeceras.js, actividades.js* y *comentarios.js*). Por ahora los archivos historia.js y contactos.js se ha guardado directamente su información en los state de *QuienesSomosComponent.js* y *ContactoComponent.js*. La función de los *reducers* es modificar el estado de nuestra aplicación, a partir de la acción que se realice. Estos reducers se construyen con un case identificando el tipo de acción.

Action Creators

En el fichero *ActionCreators.js* se implementan las funciones (*Thunk*) y las acciones. Se crean las funciones que realizaran el "fetching". Dichas funciones tendrán una implementación basada en "*promises*" haciendo uso del dispatch(). El método dispatch() es el que permite mandar acciones a los reducers

2. Refactorización de la aplicación

Se empieza por cambiar el archivo *CampobaseComponent.js* aquí se carga el store (en el state). Las funciones mapStateToProps y mapDispatchToProps dan acceso a diferentes componentes del store junto con connect.

```
import { connect } from 'react-redux';

import { fetchExcursiones, fetchComentarios, fetchCabeceras, fetchActivid
ades } from '../redux/ActionCreators';


const mapStateToProps = state => {

  return {

    excursiones: state.excursiones,

    comentarios: state.comentarios,

    cabeceras: state.cabeceras,

    actividades: state.actividades

  }

}


const mapDispatchToProps = dispatch => ({

  fetchExcursiones: () => dispatch(fetchExcursiones()),

  fetchComentarios: () => dispatch(fetchComentarios()),

  fetchCabeceras: () => dispatch(fetchCabeceras()),

  fetchActividades: () => dispatch(fetchActividades()),

})
```

Llegados a este punto se tiene actualizado los datos en el estado, es necesario actualizar la forma en que cada uno de los componentes accede a ellos.

Ahora ya no necesitaremos los ficheros de la carpeta comun como *excursiones.js, cabeceras.js, actividades.js* y *comentarios.js.* Ahora los datos se cargan mediante la función mapStateToProps:

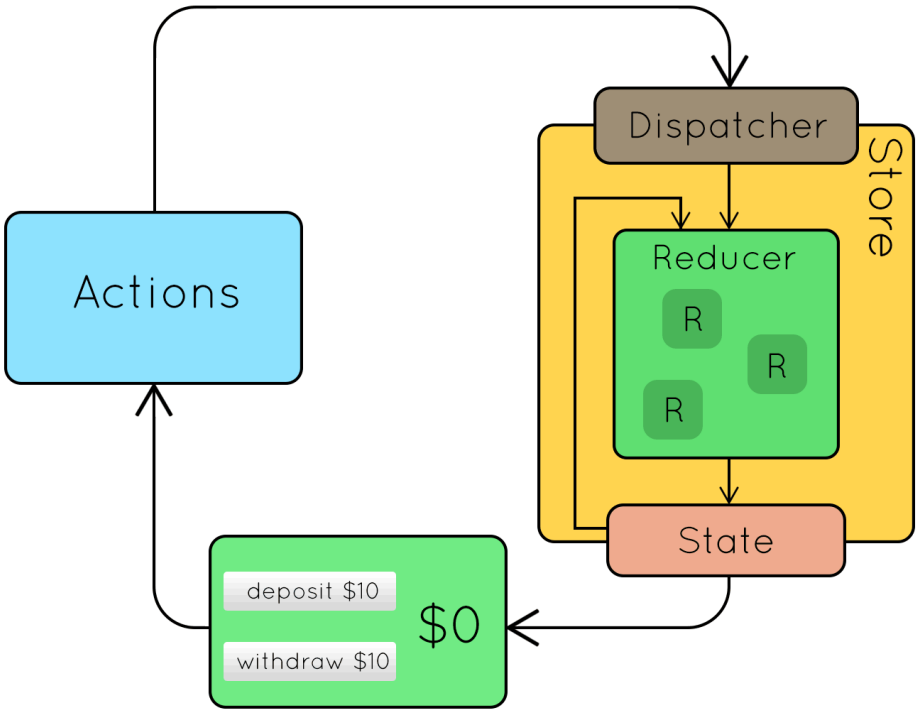
```
import { connect } from 'react-redux';

const mapStateToProps = state => {
    return {
      actividades: state.actividades,
      excursiones: state.excursiones,
      cabeceras: state.cabeceras
    }
  }

[…]
class Home extends Component {
```

```
    render() {

        return(
            <ScrollView>
                <RenderItem item={this.props.cabeceras.cabeceras.filter((
cabecera) => cabecera.destacado)[0]} />
                <RenderItem item={this.props.excursiones.excursiones.filt
er((excursion) => excursion.destacado)[0]} />
                <RenderItem item={this.props.actividades.actividades.filt
er((actividad) => actividad.destacado)[0]} />
            </ScrollView>
        );
    }
}

//export default Home;
export default connect(mapStateToProps)(Home);
```

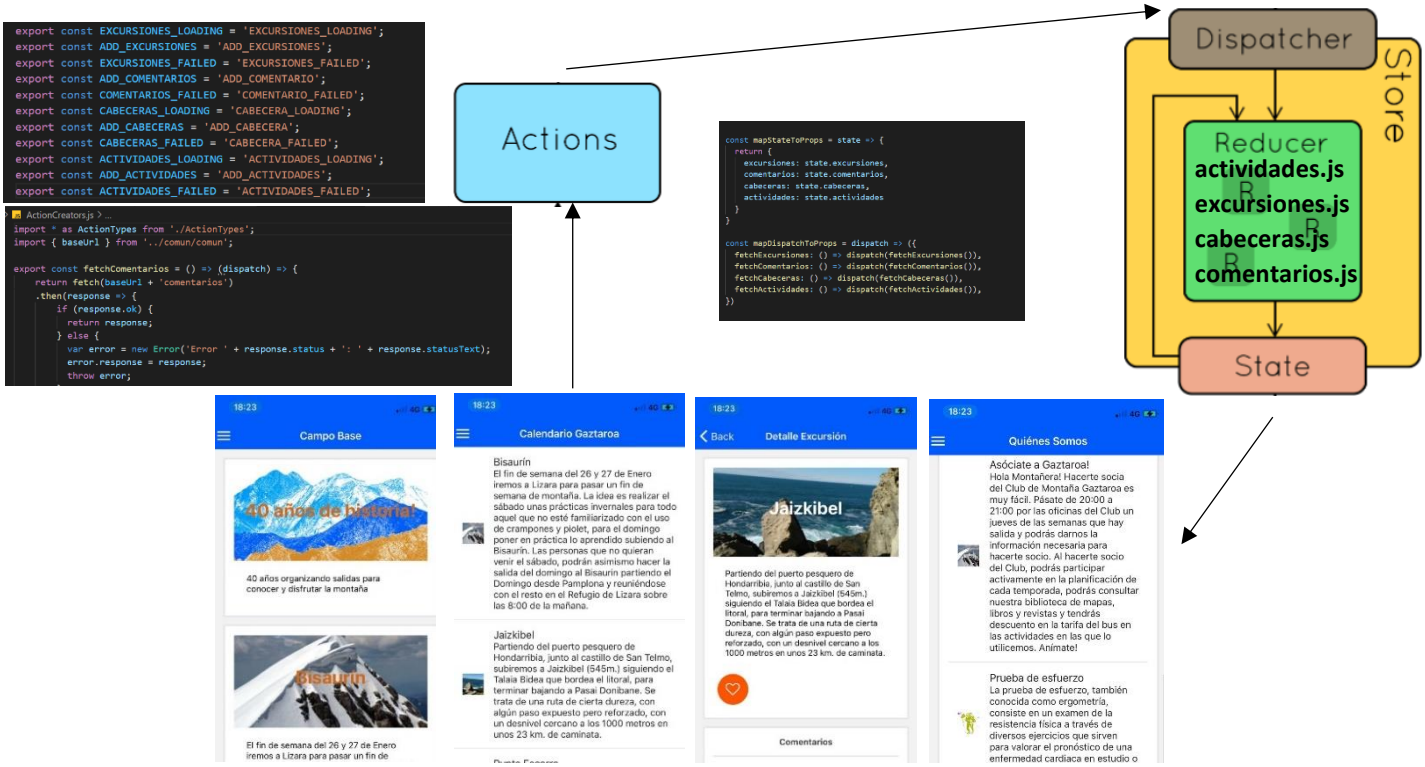
Al igual que se hace con el componente Home se debe realizar en el resto de los componentes. Hay que tener en cuenta unas excepciones por el momento el contenido de contacto y de la historia del club se guarda directamente en el estado de sus respectivos componentes pero posteriormente se tiene intención de introducirlos al db.json al igual que el resto. En DetallesComponent.js se mantiene el estado ya que de momento favoritos[], se obtendrá de esta forma.

Por último, también debemos modificar el fichero *App.js* para conectarlo a nuestro repositorio Redux.

Para entender lo que Redux está realizando en nuestra aplicación, la siguiente imagen lo explica muy bien. Cuando se produce un evento en nuestra app bien sea al clicar un botón o al clicar para ir a otro componente del menú, este evento activa una acción que llegará a nuestro store que tendrá almacenado los 4 reducers (excursiones, cabeceras, comentarios y actividades) y se carga el state correspondiente que será mandado a nuestra app.



En nuestro caso sería algo así:



Se tiene los diferentes tipos de acciones y los action creators que en función del tipo de acción devuelve una cosa u otra. Se manda la acción. El store se configura combinando los 4 reducers y con los dos middlewares.

```
import {createStore, combineReducers, applyMiddleware} from 'redux';
import thunk from 'redux-thunk';
import logger from 'redux-logger';
import { excursiones } from './excursiones';
import { comentarios } from './comentarios';
import { cabeceras } from './cabeceras';
import { actividades } from './actividades';

export const ConfigureStore = () => {
    const store = createStore(
        combineReducers({
            excursiones,
            comentarios,
            cabeceras,
            actividades
        }),
        applyMiddleware(thunk, logger)
    );

    return store;
}
```

Cargamos en el *Store* (en el *state*) los datos desde la interfaz REST API empleando las funciones de *"fetching"*. Esta operación se realiza en el fichero *CampoBaseComponent.js.* Finalmente los datos estan disponibles inmediatamente después de cargarse la aplicación.